DESIGN-README

The application's design follows the MVC pattern, and is divided into three modules: Model, View, and Controller.
Also, note that the application currently supports only a single user. So, the MVC entities are designed by keeping this constraint in mind.

Design Changes for Assignment 6 (Stocks Part 3):-

- Assignment 6 introduces new strategy features for investment and GUI view.
- **Features**
    - We have added a new interface named **Features** that lists all the features/ functionalities our application supports. The purpose of having this new interface is that the controller won't have to implement ActionListener.
- **GUIController**
    - This controller class implements the Features interface.
    - The new controller class is introduced to communicate with the new view class and existing model class.
    - The controller takes inputs from the view and passes these inputs to the model to perform the required operations.
- **IGUIView**
    - IGUIView is the new interface that has methods to implement the GUI interface.
    - This interface has methods to take inputs from the user, and show components on the screen.
    - **GUIView** class implements the IGUIView interface
- Model changes
    - We have added new methods to the existing model to support the new investment strategy features.

Design Changes for Assignment 5 (Stocks Part 2):-

- Assignment 5 introduces new features/ operations to perform on a portfolio. The following changes are made in order to incorporate these features:-
    - **Sell stocks from a portfolio**
        - We've created a new method in the **IModel** interface that sells the stocks from a portfolio.

        - The above method basically deletes the stocks from the **IStock** list (a list of stocks maintained in the **Stock** class for each portfolio) of that particular portfolio.
    - **Commission fee for transactions**
        - We've introduced a new method in **IModel** that sets the commission fee given by the user.
        - After each transaction (creating a portfolio, adding a stock to the portfolio, selling stocks) we are reducing the commission fee by a certain percentage which is user input.
    - **Portfolio performance over time**
        - A new interface **IPortfolioPerformanceData** is introduced to represent the performance of data over time. This interface has methods to get the performance-related information of a portfolio.
        - **PortfolioPerformanceData** class implements the **IPortfolioPerformanceData** interface.
    - **API integration**
        - In the previous implementation, we called the API and populated the offline, internal database with stock values.
        - In this assignment, we have completely integrated the API, and instead of using the local database to calculate the portfolio's value, we are relying on the API to fetch stock values.
    - New methods to display the appropriate message for each feature have been created in the **IView** interface.
    - Created an enum **StockDataSource** that contains the list of all data sources supported by the program to get the portfolio value.
    - The switch cases in **PortoflioController** class are moved to private helper methods to make the code more readable.

Application's design:

- **Controller**
    - The Controller is the entity that acts as the middleware between the View and Model. It is essentially the driver of the application.
    - The Controller is responsible for calling the View's methods to take inputs from the user and display messages/errors.
    - The Controller then delegates the task that needs to be performed on the user input by relaying the user inputs to the Model.

- The Controller is not responsible for determining how the Model processes the user data, or how the Model is implemented internally. The Controller also does not care about how the View takes user inputs or displays the data/results to the user. This way, the Controller achieves loose coupling, and any Model or View can be plugged into the Controller without breaking the application.
- The Controller provides a single and only method for starting the application, which marks the start of the program.
- This application's Controller provides a method named "start()" which, when called, starts the application.
- The Controller is specified by the **IController** interface and is implemented by the **PortfolioController** class.

- **View**
  - The View is the entity that is responsible for taking care of the application's interaction with the user.
  - The View is responsible for taking user inputs and also displaying outputs to the user. The View does this by providing methods through which it can provide functionality for prompting users to give input, or displaying some result based on user input.
  - This application's View provides various methods for providing the user with a menu of options for creating a portfolio, examining a portfolio, getting the value of a portfolio, displaying the supported stocks in the application, selling stocks from the portfolio, adding stocks to an existing portfolio and checking the portfolio's performance over time.
  - The View also handles various error or exception scenarios by providing the user with appropriate messages and providing the next available steps.
  - The View is specified by the **IView** interface and is implemented by the **PortfolioView** class.

- **Model**
  - The Model entity is responsible for executing the core logic of the application. The Controller is simply an agent that calls the Model's methods and then the Model performs the appropriate actions using the underlying logic that can vary from one implementation of the Model to another.

- The Model of this application is responsible for providing the functionality for creating a portfolio, examining a portfolio, getting the value of a portfolio, displaying the supported stocks in the application, and more.
- The Model performs by assuming that there is a single user of the application as of now.
- The Model is specified by the **IModel** interface and is implemented by the **PortfolioModel** class.
- **PortfolioModel**
  - The **PortfolioModel** class internally has a list of interface type **IPortfolio**, the list of all portfolios created by the user, and a data store of stock data defined by the interface **IStockDataStore**.
  - This class contains the main methods of the application, other internal functionality of the application is hidden from the user.
  - This class stores the portfolios created by the user in the application's memory in the form of an object and avoids inefficiencies caused by reading files. Also, the newly created portfolios are stored in local storage as CSV files, if there is a need to perform additional actions in the future.
- **IPortfolio**
  - The **IPortfolio** interface defines a stock portfolio.
  - A portfolio may have basic properties like a portfolio name and a list of stocks.
  - This interface is implemented by the **Portfolio** class.

- **Portfolio**
  - The **Portfolio** class has a list of stocks. A stock is represented by the **IStock** interface.
- **IStock**
  - The **IStock** interface represents a stock in the stock portfolio. A stock has properties like stock name, stock quantity, and stock purchase date.
  - This interface is currently implemented by the **Stock** class.
- **IStockDataStore**
  - The **IStockDataStore** interface provides methods for querying stock-related data from a stock database.
  - This interface has methods for getting a stock's value on a given date, and also for getting a list of supported stocks in the data store.

        - This interface is implemented by the **StockDataStore** class.
    - **StockDataStore**
        - This class maintains a data repository of stock for a limited number of supported stocks. This list of supported stocks is stored in local storage, so it can be easily reconfigured. The repository contains the stock prices of a stock for a wide range of dates in the stock market.
    - **FileUtility**
        - This class is a final class which is a helper class used by the **PortfolioModel**.
        - This class provides static methods for performing various file I/O operations and also operations related to a portfolio, such as reading a CSV file as a portfolio or writing a portfolio to a file.

- **Application Starter**
    - The application starter is simply a class that contains the “main()” method for starting the application.
    - This class initializes the Controller, View, and Model, and then calls the Controller’s method for starting the application.